Netsukuku topology

http://netsukuku.freaknet.org
AlpT (@freaknet.org)

February 3, 2008

Abstract

In this document, we describe the fractal structure of the Netsukuku topology. Moreover, we show how it is possible to use the QSPN v2 on the high levels of the fractal.

Contents

1 Preface

We're assuming that you already know the basics of the QSPN. If not, read the QSPN document first: [1].

2 The general idea

The aim of Netsukuku is to be a (physical) scalable mesh network, completely distributed and decentralised, anonymous and autonomous.

The software, which must be executed by every node of the net, has to be unobtrusive. It has to use very few CPU and memory resources, in this way it will be possible to run it inside low-performance computers, like Access Points, embedded devices and old computers.

If this requirements are met, Netsukuku can be easily used to build a worldwide distributed, anonymous and not controlled network, separated from the Internet, without the support of any servers, ISPs or control authorities.

3 Basic definitions

Node We call *node* any computer that is hooked up to the Netsukuku network.

Rnode stands for remote node: given a node X, it is any other node directly linked to X, i.e. it's a neighbour of X.

Map A map is a file, kept by each node, which contains all the necessary information about the network, f.e. routes and nodes status.

Example:
A is the rnode of B.

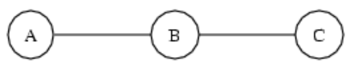

Figure 1: The nodes A,B and C

B is the rnode of A and C.
C is the rnode of B.

4 Network topology

A simple topology, which doesn't impose any structure on the network, can be memorised with a simple map. In this map, all the information regarding the nodes of the network have to be memorised. Surely, this kind of map cannot be utilised by Netsukuku, because it would require too much memory. For example, even if we store just one route to reach one node and even if this route costs one byte, we would need 1Gb of memory for a network composed by 10^9 nodes (the current Internet).

For this reason, it's necessary to structure the network in a convenient topology.

4.1 Fractal topology

4.1.1 Level 1

First of all we'll subdivide the network in groups of 256 nodes and we'll use the following definitions:

Gnode means group node. It is a group of nodes, i.e. a set of nodes. Each node of the network belongs to just one gnode.
A gnode contains a maximum of 256 nodes.
By writing $n \in G$ we mean that the node n belongs to the gnode G.

Bnode stands for border node. It is a node which belongs to a gnode G, but that is also directly linked to at least one node of another gnode, i.e. some of its rnodes belongs to different gnodes than its.
By writing $b \in G$ we mean that the bnode b belongs to the gnode G.

Example:
$A \in G$, A is a node belonging to the gnode G, its rnode is B.

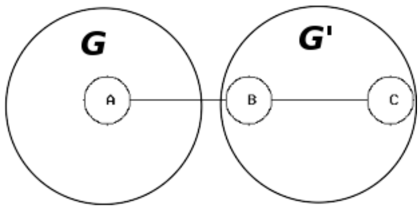

Figure 2: The bnode A and B, belonging respectively to the gnode G and G'

$B \in G'$, B is a node belonging to the gnode G', its rnode is A.
A is a bnode of G, while B is a bnode of G'.

4.1.2 Level n

We further subdivide the network topology in *groups of 256 groups of nodes* and we continue to name them as gnode.
At this point, we repeat recursively this subdivision process until we can group all the nodes of the network into a single gnode.

Doing so, we've structured the network in $n+1$ levels (from 0 to n).
In the base level (level 0), there are 256 single nodes.
In the first level (level 1), there are 256 normal gnodes. Each of them contains 256 single nodes.
In the second (level 2), 256 gnodes of level 1 forms a single *group of groups of nodes.*
In the third (level 3), there are 256 groups of 256 groups of 256 groups of 256 nodes.
Continuing in this way, we arrive at the last level (level n), where there is a single group which contains the whole network.

The QSPN algorithm is able to operate independently on any level, considering each gnode as a single node of level 0. For this reason, we can view the Netsukuku topology as a fractal, where each level is composed by single nodes.

Example

Figure 3[1] is an example of the fractal topology of Netsukuku.

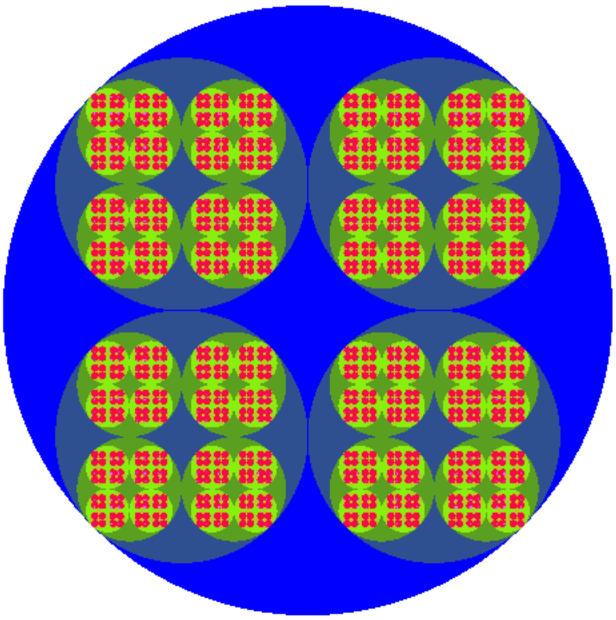

Figure 3: An example of the netsukuku topology structure

In this topology, each gnode contains four nodes, i.e. each group contains four elements. The network is structured in 6 levels.
The red elements, are single nodes (level 0).
Four nodes forms a single group of nodes (level 1).
A single bright green circle is a group of groups of nodes (level 2).
The dark green circles are groups of groups of groups of nodes (level 3).
The dark blue circle are groups of groups of groups of groups of nodes (level 4).
Finally, the bright blue circle is the gnode which contains the whole network (level 5).

4.1.3 Membership

Let's assign a numeric ID to each (g)gnode, starting from the last level:

1. in the last level (n) there's only one giant gnode, thus we assign to it the ID "0". Our global ID will be:

$$0$$

[1]this figure has been taken from: http://www.ian.org/FX/Plugins.html

2. in $n-1$ there are 256 gnodes, which belongs to the gnode 0 of level n, thus we assign them the IDs from 0 to 255. The global ID becomes:

$$0 \cdot i \quad 0 \leq i \leq 255$$

3. we repeat the step 2 recursively gaining an ID of this form:

$$0 \cdot i_{n-1} \cdot i_{n-2} \cdot \cdots \cdot i_0 \quad 0 \leq i_j \leq 255, \ 0 \leq j \leq n-1$$

4. since the last level is always 0, we'll omit it and we'll consider only the first n levels.

In a network with a maximum of 2^{32} nodes (the maximum allowed by the ipv4), there would be five levels ($n = 4$), where each gnode will be composed by 256 nodes. Therefore, the ID will be in the usual IP form:

$$0 \ldots 255 \cdot 0 \ldots 255 \cdot 0 \ldots 255 \cdot 0 \ldots 255$$

For example, a single node of level 0 of the network is:

$$3 \cdot 41 \cdot 5 \cdot 12$$

That said, each gnode of the network belongs to only one combination of gnodes of the various levels. In our previous example we have:

$$\begin{aligned} g_3 &= 3 \\ g_2 &= 41 \\ g_1 &= 5 \\ g_0 &= 12 \end{aligned}$$

where each g_i corresponds to the gnode ID of the level i. Note that g_0 is the ID attributed to the single node, at level 0.

4.2 Fractal map

The advantages of using a fractal topology are clear.
The node N, instead of memorising information about each node of the whole network, will keep only that regarding the gnodes where it belongs to. Suppose the node N had this ID:

$$g_3 \cdot g_2 \cdot g_1 \cdot g_0$$

It will store in memory information regarding:

1. the 256 single nodes which belongs to its same gnode of level 1, or in other words, the 256 nodes of the gnode g_1,

2. the 256 gnodes gnodes which belongs to its same gnode of level 2, of in other words, the 256 gnodes of the gnode g_2,

3. finally, the 256 gnodes which belongs to the gnode g_3.

Note that doing so, the node N will be blind to all the other gnodes. For example, it won't know any information regarding the single nodes which belong to all the other gnodes of level 1 different from g_1.

Even with this lack of knowledge, as we'll see later, the node N is still able to reach all the other nodes of the network. In conclusion, N only needs $256n$ entries in its map, instead of 2^{32}. To clarify the ideas suppose that each entry costs one byte. In the plain topology we needed $4Gb$, while in the fractal one we just need $256 \cdot 4\ b = 1Kb$.

4.2.1 IP v4 and v6

Netsukuku is both compatible with ipv4 and ipv6.

In ipv4 there are a maximum of 2^{32} IPs, thus we have five levels $n = 4$.
In ipv6 there are a maximum of 2^{128} IPs, thus $n = 16$.

4.2.2 Internal and external map

For simplicity we divide the map of the node N, in the *internal map* and in the *external* one. The internal map contains information regarding the nodes belonging to g_1. The external map describes all the other levels of the topology.

4.2.3 Bnode map

The bnode map of the node N contains the information regarding the bnodes of each level where N belongs. Some examples to clarify the ideas:

suppose that $N = g_3 \cdot g_2 \cdot g_1 \cdot g_0$

- a bnode of level 0 is a single node linked with two nodes of two different gnodes of level 1.
- the bnodes of level 0, known by N, are only that which belong to the gnode g_0. They are all the nodes of g_0 which are linked to at least a gnode different from g_1.

4.3 CIDR routing

The QSPN, for each level, will build the routes necessary to connect each (g)node to all the other (g)nodes of the same level. The routes will be saved in the maps of each node.

If the node $N = g_3 \cdot g_2 \cdot g_1 \cdot g_0$ wants to reach a node M which belongs to different gnodes, f.e. $M = g_3 \cdot g_2 \cdot h_1 \cdot h_0$, it will add a CIDR[3] route in the routing table of the kernel:
all the packets whose destination is $g_3 \cdot g_2 \cdot h_1 \cdot 0 \ldots 255$ *will be forwarded to the gateway* X.

We'll see later how the gateway X is chosen.

5 Tracer Packets in high levels

In the QSPN document [1], we've seen how a Tracer Packet works in a network composed by single nodes, i.e. a gnode of level 0.
We'll now study its way of working on higher levels.

5.1 A gnode is a node

In the abstract sense a single node is an entity which:

1. receives input from its links
2. stores it in its memory
3. computes it
4. and sends the output of the computation over some of its links

Thus any other entity which performs the same operations can be thought as a single node.
A gnode G can act as a single node too.

1. A bnode I, which belongs to G, receives an input from its links. We call I the *ingress* (b)node.
2. this input is flooded to all the nodes, of any level, of the gnode. The nodes will memorize the information contained in the input.
   Note that the flood is not a TP. The flooded pkt will be received only once by each node.
3. A bnode O, of the same gnode, which is different from I, receives the flooded input and computes it. We call O the *egress* (b)node.
4. The bnode O sends the output of its computation to its external links, i.e. avoiding those links which connect it to nodes of G or to the same gnode which sent the input to I.

5.1.1 Example of a wandering TP

Consider the network in figure 4.
G_1, G_2, G_3 are gnodes of level 1. A_1 and A_2 belong to G_1. B_1 and B_2 belong to G_2. C_1 belongs to G_3.

Suppose the node A_1 sends a Tracer Packet in level 1 [2]. The following will happen:

1. The TP is flooded in G_1.
2. A_2 receives the TP and appends in it the ID of the gnode G_1.
3. A_2 sends the TP to B_1.
4. B_1 receives it, updates its maps and floods it in G_2.

[2] Note that any node can send a TP in any level. In this case, we are considering A_1, which is a bnode, to simplify the example

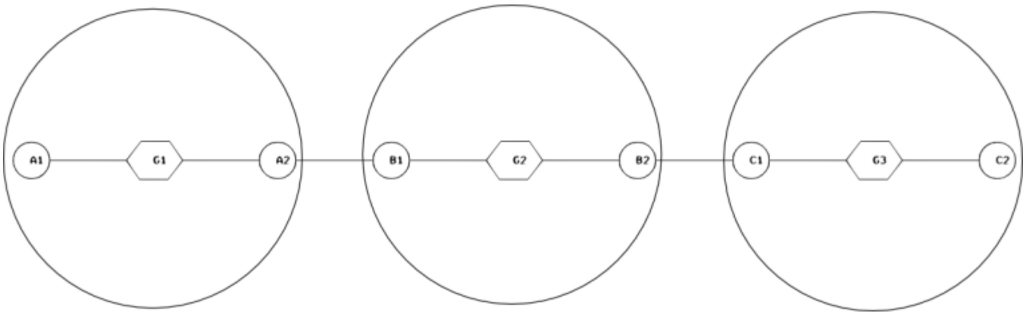

Figure 4: The gnodes G_1, G_2 and G_3

5. All the node belonging to G_2 will receive the TP, updating their maps.

6. This same procedure is reiterated from step 2, i.e. B_2 receives the TP, appends the ID of G_2 and so on until C_1 receives it.

7. C_1, noticing that its gnode hasn't any links to other gnodes than G_2, will bounce back the TP to B_2 and at the same time will flood G_3.

8. The TP, with the same procedure, will return back to A_1, completing the TP cycle.

6 QSPN v2 in high levels

In order to use the Q^2 (QSPN v2) in high levels, we need to be sure that a TP, flooded inside a lower gnode, will reach once and only once all the nodes of the same gnode. Moreover, a good metric needs to be defined for the high levels: what is the rtt (Round-Trip Time) and bandwidth capacity of the link between two gnodes, how is it measured?

6.1 Endless loops

Consider the situation in figure 5. A and B are bnodes of the node G_1, while C

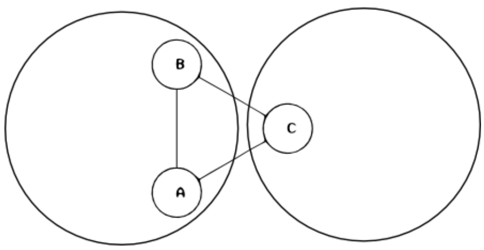

Figure 5: Three bnodes, forming a cycle

is a bnode of the gnode G_2. Suppose C sends a TP to B. B floods it inside G_1, thus A receives it. At this point, A sends the packet to all its external links,

and thus to C. C will send the packet again to B and the cycle will continue.

This situation can be avoided if an ingress bnode, before forwarding a packet to a an external gnode F, checks if the packet has already traversed F itself. If it has, the bnode won't forward it to F.

6.2 Route Efficiency Measure

We will refer to REM (Route Efficiency Measure), as a value characterising the quality of a route. REM can be calculated in various ways, f.e. by taking in account the total rtt and the bw capacity of the route. We denote the REM of a route r as $REM(r)$.

6.3 Flat levels

From the point of view of the QSPN v2, the levels are "flattened", because the propagation of CTPs[3] ignore the subdivisions of the network in gnodes and levels. The structure of the network remains unchanged, subdvided fractally in gnodes, but the exploration of the CTP doesn't. A CTP doesn't consider a gnode as a single, homogenous entity. It propagates itself between the single nodes, tracing a precise route formed by single hops, until it is considered interesting. In this way every node will have its own personal route to reach a specific gnode.

The memory limits are respected, because the CTP, once it jumped over the border of a gnode, deletes the specific details that were useful by the nodes of the gnode.

We'll know describe the rules of the Flat levels.
Suppose we want to explore the level $n \geq 1$ (we already know how to explore $n = 0$). Let $G_1, \ldots, G_r$ be the gnode of level n. The generic rule of the QSPN v2 in high levels is:

> When an ingress bnode b of G receives a CTP from a neighbouring bnode of H, it appends in the packet its IP. The packet, called *Locked Tracer Packet* is then flooded inside G. The LTP of level n, is a packet that is propagated with the same rules of the CTP, i.e if it isn't interesting it's dropped, but that isn't updated while it explores the levels inferior to n. When the egress bnode b', of G receives the LTP, it changes it in a CTP, appends the ID of G and forwards it to all its neighbouring gnodes.

Detailed rules:

1. When exploring the level n, each inferior level must been already explored. To accomplish this, it sufficies that a node, before sending/forwarding a CTP of level n has already sent/forwarded at least a CTP of level $n - 1$. If it hasn't, it will queue the CTP of level n.

2. Suppose that the ingress bnode b belonging to a gnode G of level n, has received a CTP of the same level, sent by the gnode G'. The following

[3] Continuous Tracer Packet, see [1]

will happen: b makes copy of the CTP, marking it as a LTP, appends in it its IP and floods it in G. If b is also linked to a gnode different from G', it will respect the rule 3 for egress bnodes.

3. Suppose that the egress bnode b' of G receives the LTP sent by b (see above). b' transforms the LTP in a CTP, deletes the IP added by b and sets (in the TP) the REM of $G \to G'$ as:

$$REM(G \to G') := REM(b' \to b)$$

Finally, it forwards the CTP to all its external rnodes.

4. A node n of G, uses the received LTP to learn that b is a bnode bordering on G' and to find out the route of level n contained in the packet. It considers the REM of the route to reach G' equal to

$$REM(n \to b)$$

in other words, it saves in its map that $REM(n \to G) := REM(n \to b)$. n forwards the LTP to its rnodes without modifying it.

5. The LTP respects the same rule of the QSPN v2: it is forwarded by a node only if it is carries interesting information, otherwise it is dropped. A node n consider the LTP interesting if one of the following conditions[4] is met:

   - The LTP contains the IP of a bnode b, which wasn't already known by n.
   - The LTP contains a bnode already known by n, but that borders on a new gnode g: with this LTP, the node n learns that it can reach g by passing through the bnode.
   - The LTP contains the information of the death of a bnode which was used by n to reach some gnodes.
   - The LTP contains a new gnode, which was previously unknown.
   - The LTP contains the information of the death of a gnode.
   - The LTP contains an improved route to reach a gnode.

   Note: the first three conditions are used by the nodes to build and update the bnode map.

6. Recall the situation of rule 3.
   When the value $REM(b' \to b)$ changes considerably, both b and b' sends a CTP of level n to their external nodes[5]. The CTP contains the new REM value. Every node which used G as hop of some routes, will consider the CTP interesting.
   This rule permits the update of high levels.

[4]this isn't the complete list of conditions

[5]A CTP of level $n-1$ tells b and b' of the REM change

7. When a bnode b (of any level)[6] loses one of its external links of level n, a CTP is sent in the level $n-1$. The CTP contains the information regarding the lost link. The CTP is forwarded by all the (g)nodes that used the bnode b in one of their saved routes to reach the lost link.
   If b was the unique bnode of its gnode G bordering to gnode G' of level n, then the link $G \leftrightarrow G'$ becomes broken. In this case a CTP of level n is sent by G' and by the other bnodes of G[7].
   A dying bnode is equivalent to a bnode which loses all its external links. The only difference is that a CTP of level 0 is sent to inform the nodes of its gnode.

We'll now describe some rules by giving an example (see figure 6). For simplicity, we'll use the IP notation: 11.22.33 indicates the single node 33 belonging to the gnode 22, belonging to the ggnode 11. Using the symbolic notation: $11.22.33 \in 11.22 \in 11$. We are also assuming that the routes are symmetric.

Figure 6: Level 3

CASE 1: A new gnode join.
We are exploring the level 3, which is formed by the gnodes 11, 22, 33, 44. The level 2, of each gnode, has been already explored.

- The gnode 44 has just created the link to the gnode 33. The new physical link is maintained by the two single bnodes 33.33.33, 44.44.44.
- The bnode 33.33.33 creates a CTP of level 3, appends in it the ID "33" and sends it to 44.44.44.
- The bnode 44.44.44 receives it.
  - It reflects back a new CTP to 33.33.33, appending in it the ID "44".
  - It appends in the received CTP, its IP, marks it as a LTP and sends it to its gnode. The LTP contains the path $33 \to 44.44.44$.
    * The node $44.44.x$, receiving the LTP, learns the following: 44.44 is a bnode of level 2 that can be used to reach 33, and 44.44.44 is a bnode of its own gnode which borders on 33. It also learns its own REM for the route to 33:

      $$REM(44.44.x \to 33) := REM(44.44.x \to 44.44.44)$$

      $44.44.x$ continues to forward the LTP (without modifying it).
    * The LTP reaches the node $44.y.z$, node of $44.y$. $44.y.z$, from the LTP learns that: 44.44 is a bnode of level 2 that can be used to reach 33. Since the level 2 has been already explored, $44.y.z$ already knows a route to reach 44.44.

[6] a bnode b'', which is a gnode of level $l \geq 1$, loses its external link to the gnode H of level $l+1$, when all its bnodes of level $l-1$ loses their link to H. This is a recursive definition.
[7] they will know of the broken link when receiving the CTP of level $n-1$

    * The LTP having reached all the nodes of 44 ceases to being forwarded.

- The bnode 33.33.33 receives the reflected CTP sent by 44.44.44.
  - It appends in it its IP, marks it as a LTP and sends it to its gnode.
    * The LTP is propagated in 33 (in the same way of the previous LTP).
- The bnode $33.r.s$, which borders on 22, receives the LTP.
  - It modifies it in a CTP: it removes the IP 33.33.33 and appends the ID 33. The CTP is now $44 \rightarrow 33$. Finally, it sets the first REM of the CTP:

    $$REM_{CTP}(33 \leftrightarrow 44) := REM(33.r.s \leftrightarrow 33.33) \quad \text{see note [8]}$$

    The CTP is then sent to 22.22.22, bnode of 22.
- The bnode 22.22.22 receives the CTP. It sends a LTP in 22.
- The bnode $22.t.u$, which borders on 11, receives the LTP. It appends 22 in it and sends it to 11. The CTP is now: $44 \rightarrow 33 \rightarrow 22$.
- A bnode of 11 receives it, and reflect it to 22. It sends also the LTP to 11.
- ...
- The reflected CTP is finally received by the node 44.44.44, which sends the LTP to 44 and reflects the CTP to 33.33.33. The bnode 33.33.33, drops immediately the CTP, because it doesn't contain any interesting information.
- the exploration is completed.

6.4 Unique flood

Suppose that a TP has been flooded inside the gnode G. Suppose also that the node $n \in G$ receives two duplicate packets.
The Q^2 instructs the node n to keep forwarding only the interesting packets. A packet, which is a perfect copy of a packet already received, is always uninteresting. Therefore, the node n will drop the second copy of the received TP.

However, in this case the node n should not consider the REM values saved in the TP, because two TPs, which have crossed the same gnodes in the same order, might have different REM values.

In conclusion, whenever two or more TPs, which have crossed the same high level route, are received by a node of G, only one of them will be forwarded. In this way, the successive nodes will receive only one copy of the same packet.

[8]The node $33.r.s$ knows that 33.33 is a bnode of level 2 used to reach 44 because it is written in the LTP. Moreover it knows a route to reach 33.33, the level 2 has been already explored. Thus $REM(33.r.s \leftrightarrow 33.33)$ is a known measure.

7 Network dynamics

When a part of the network changes considerably, the maps of the involved levels must be updated.

7.1 Radar

Every node has its own radar, which periodically sends a broadcast request to all its (physical) near nodes. By collecting the replies, the radar is able to determine the active rnodes of the node and the quality of its links.

7.2 Level 0

In level 0, a CTP is sent every time a node joins the network, dies or every time the change of the quality of a link exceeds a predefined delta value. The CTP is also is restricted to the gnode where it has been originated: a bnode won't send it to external nodes.

When a node joins, it won't send a CTP, only its rnodes will. Their CTP will be directed to the node. More formally:
if the node n joins, that is if the node n creates a link with the nodes $r_1, \dots, r_n$, with $n \geq 1$, then a CTP will be sent by each rnode r_i *only* to n. This means that if r_j is connected to a node s, when n joins, r_j will send the CTP only to n and not to s.
This saves the propagation of two CTP in the following situation:

$$\dots \leftrightarrow A \leftrightarrow B \leftrightarrow C \leftrightarrow \dots$$

suppose that the node B joins. As a consequence of the rule we've described, the node C will send a CTP to B and A to B, thus only two distinct CTP are generated. Their path is $A \to B \to C \to \dots$ and the reverse. If instead the node B sends a CTP to C and then A to B, two distinct CTP will explore the same verse: $B \to C \to \dots$ and $A \to B \to C \to \dots$, the same is for the reverse, thus, in total, four distinct CTP are propagated.

Let $A \overset{l}{\leftrightarrow} B$ be a link. If its quality changes B will send a CTP to A and vice-versa.

When a node dies, all its rnodes will send a CTP to all their rnodes. The CTP will include, as the first hop, the ID of the dead node, with a flag which indicates its death. For example:

$$A \leftrightarrow B \leftrightarrow C \leftrightarrow \dots$$

if A dies, then B will send the following CTP to C: $\overset{+}{A} \to B$

In order to prevent false positives, the nodes won't immediately send the CTP, but will wait a small amount of time. Only if the change persists, they will send it.

7.3 Level n

The dynamics for the update of high levels are mainly governed by rule 6 and 7 of flat levels (see 6.3). In this paragraph, we'll specify some details.

Rule 6 says: "when the value $REM(b' \to b)$ changes considerably, a new CTP is sent". By considerably we mean that the difference of the current REM value with the previous one exceedes a predefined threshold. More precisely: let $A \to B$ be a route of level n. If

$$|REM_{\text{now}}(A \to B) - REM_{prev}(A \to B)| > \Delta_n$$

then the route has changed considerably. Δ_n is proportional to n, in this way the nodes will be more sensible to minor changes of lower levels and they will consider only relevant changes of higher levels.

Also in high levels, a bnode won't immediately send a new CTP, but it will delay it for a small amount of time.

7.4 Hooking phase

A new node joins the network when it has been able to create at least one physical link to an active Netsukuku node and when it has correctly executed the hooking procedure. In this paragraph, we describe loosely the hooking phase of a new node.

Suppose that the node n has established a physical link to at least one Netsukuku node. In order to become an active Netsukuku node, n has to *hook* to its rnodes.

During the hook, n will exchange vital information with its rnodes, it will choose its new IP and it will finally become part of a gnode.

The hook procedure is formed by these general steps:

1. The node n chooses an IP in the range of $10.0.0.1 \leq IP \leq 10.0.0.255$.
2. It launches the first radar to see what its rnodes are. If not a single node is found, it creates a new gnode and ends hooking phase.
3. At this point, n asks to its nearer rnode the list of all the available free nodes presents inside the gnode of the rnode.
   If the rnode rejects the request (the gnode might be full), the node n contacts another rnode.
4. n chooses an IP from the received list of free nodes and sets it on its network interface.
5. n will then download the external map from the same rnode. Looking at the external map, it will be able to determine if it has to create a new gnode. If it has, it creates it and ends the hooking.
6. n gets the internal and the bnode map from the same rnode.
7. n launches a second radar and updates its routing table.
8. All the rnodes of n send a CTP to update the maps.

7.5 Gnode hook

When a node creates a new gnode, it will choose a random gnode ID, and thus a random ip.
Suppose that two isolated gnodes get the same gnode ID. When they will be linked, they'll enter in conflict.
The solution to this problem is to let each new gnode hook as a normal node would. You can find more information about this in the NTK_RFC 001[4].

8 ChangeLog

- `March 2007`
  - Description of the Flat levels (sec. 6.3)
  - Section 7.2 "Network dynamics - Level 0" expanded.
  - Section 7.3 "Network dynamics - Level n" updated: the references to the pre-Flatlevels REM metric have been removed.
- `October 2006`
  Initial release.

References

[1] QSPN document: qspn.pdf

[2] Netsukuku website: http://netsukuku.freaknet.org/

[3] CIDR routing: Classless_Inter-Domain_Routing in Wikipedia

[4] NTK_RFC 001: Gnode contiguity

^_^